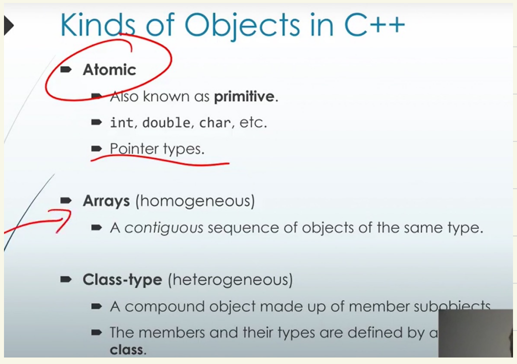
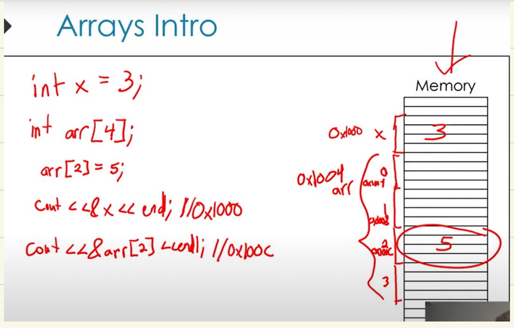
4 arrays and memory

Video one

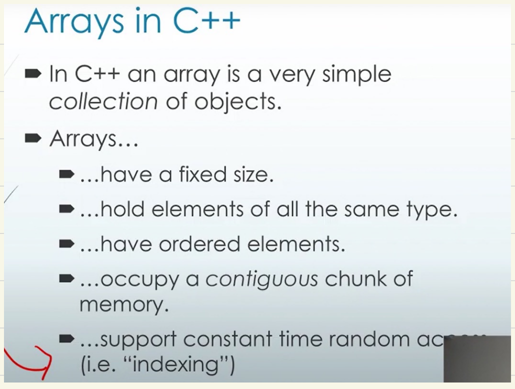


Kinds of Objects in C++

- Atomic
  - Also known as **primitive**.
  - int, double, char, etc.
  - Pointer types.
- Arrays (homogeneous)
  - A *contiguous* sequence of objects of the same type.
- Class-type (heterogeneous)
  - A compound object made up of member subobjects.
  - The members and their types are defined by a class.

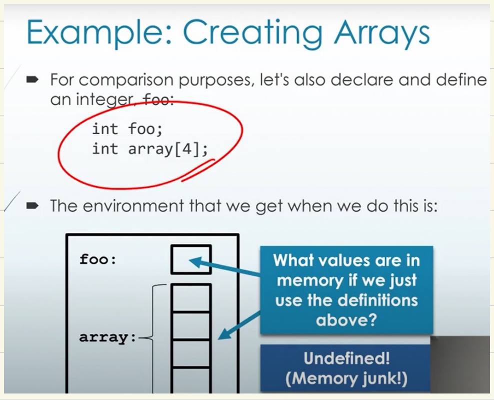


Arrays Intro



Arrays in C++

- In C++ an array is a very simple *collection* of objects.
- Arrays…
  - …have a fixed size.
  - …hold elements of all the same type.
  - …have ordered elements.
  - …occupy a *contiguous* chunk of memory.
  - …support constant time random access (i.e. "indexing")



Example: Creating Arrays

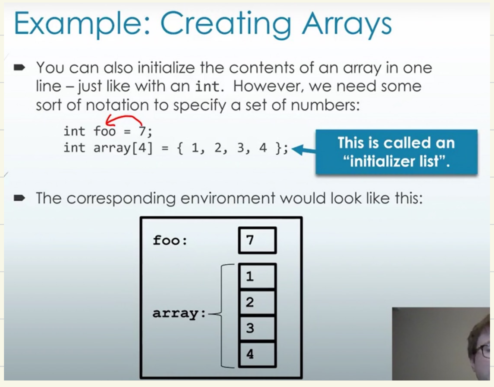
- For comparison purposes, let's also declare and define an integer, foo:

```
int foo;
int array[4];
```

- The environment that we get when we do this is:

What values are in memory if we just use the definitions above?

Undefined! (Memory junk!)



Example: Creating Arrays

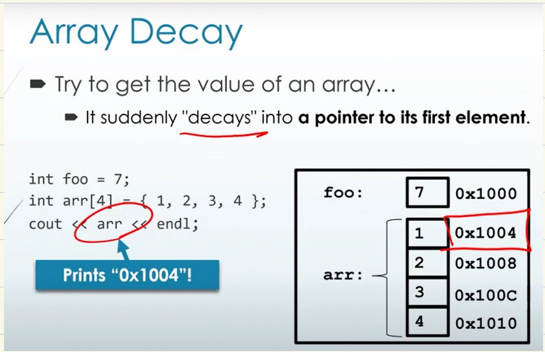
- You can also initialize the contents of an array in one line – just like with an int. However, we need some sort of notation to specify a set of numbers:

```
int foo = 7;
int array[4] = { 1, 2, 3, 4 };
```

This is called an "initializer list".

- The corresponding environment would look like this:

Video two: Arrays, pointers and pointer arithmetic

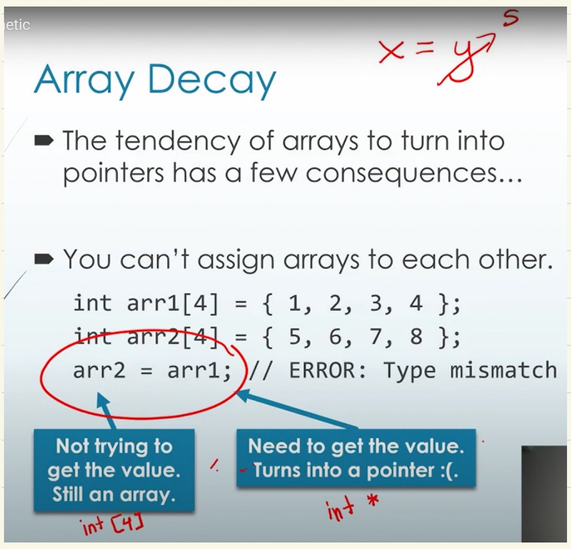


Array Decay

- Try to get the value of an array…
  - It suddenly "decays" into **a pointer to its first element**.

```
int foo = 7;
int arr[4] = { 1, 2, 3, 4 };
cout << arr << endl;
```

Prints "0x1004"!



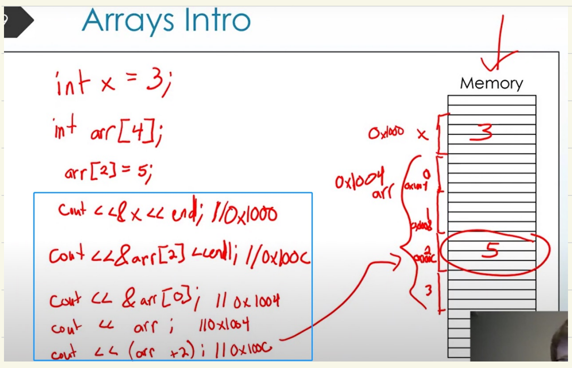
Array Decay

- The tendency of arrays to turn into pointers has a few consequences…
- You can't assign arrays to each other.

```
int arr1[4] = { 1, 2, 3, 4 };
int arr2[4] = { 5, 6, 7, 8 };
arr2 = arr1; // ERROR: Type mismatch
```

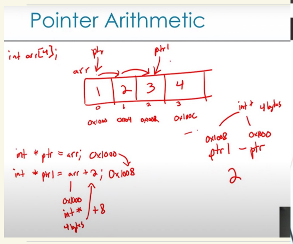
Not trying to get the value. Still an array.

Need to get the value. Turns into a pointer :(.

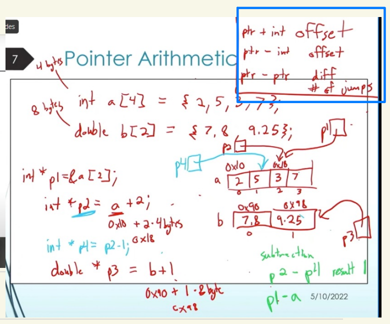


Arrays Intro

Memory
int x = 3;
int arr[4];
arr[2] = 5;
Cout << &x << endl; //0x1000
Cout <<&arr[2] <<endl; //0x100C
Cout << &arr[0]; // 0x1004
cout << arr ; //0x1004
cout << (arr +2); //0x100C
0x1000 x
0x1004 arr
3
5

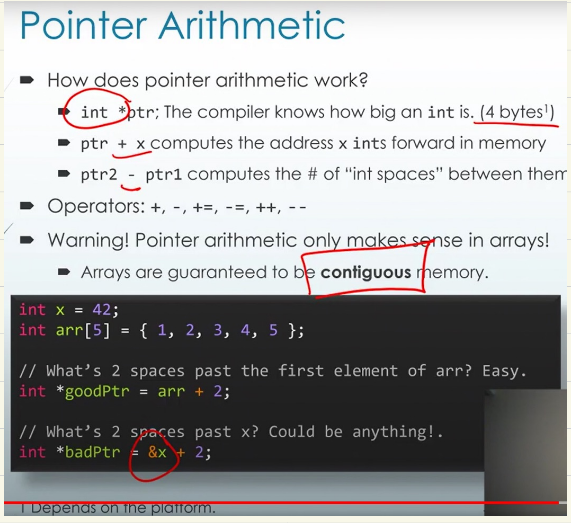


Pointer Arithmetic



7 Pointer Arithmetic

5/10/2022



Pointer Arithmetic

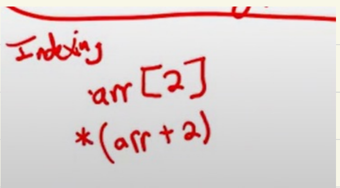
- How does pointer arithmetic work?
  - `int *ptr;` The compiler knows how big an int is. (4 bytes[1])
  - `ptr + x` computes the address x ints forward in memory
  - `ptr2 - ptr1` computes the # of "int spaces" between them
- Operators: +, -, +=, -=, ++, --
- Warning! Pointer arithmetic only makes sense in arrays!
  - Arrays are guaranteed to be **contiguous** memory.

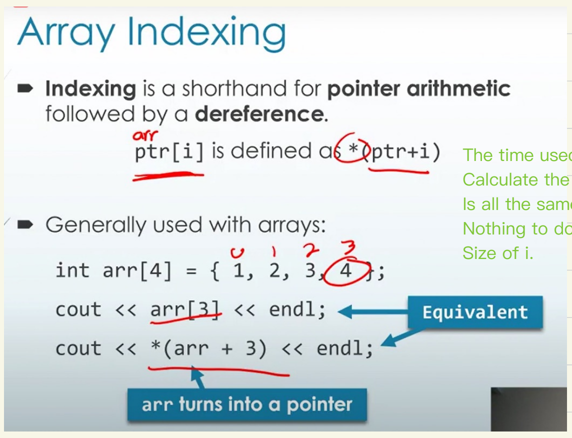
```
int x = 42;
int arr[5] = { 1, 2, 3, 4, 5 };

// What's 2 spaces past the first element of arr? Easy.
int *goodPtr = arr + 2;

// What's 2 spaces past x? Could be anything!.
int *badPtr = &x + 2;
```

[1] Depends on the platform.





Array Indexing

- **Indexing** is a shorthand for **pointer arithmetic** followed by a **dereference**.

`ptr[i]` is defined as `*(ptr+i)`

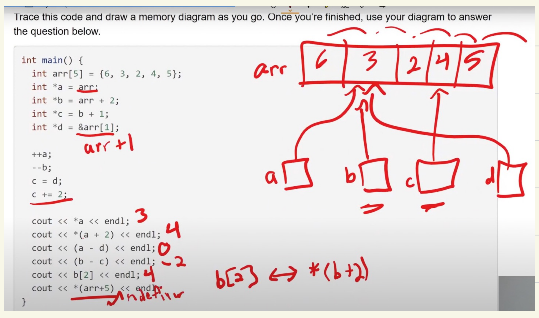
- Generally used with arrays:

```
int arr[4] = { 1, 2, 3, 4 };
cout << arr[3] << endl;
cout << *(arr + 3) << endl;
```

Equivalent

arr turns into a pointer

The time used to Calculate the ptr + i Is all the same and has Nothing to do with the Size of i.



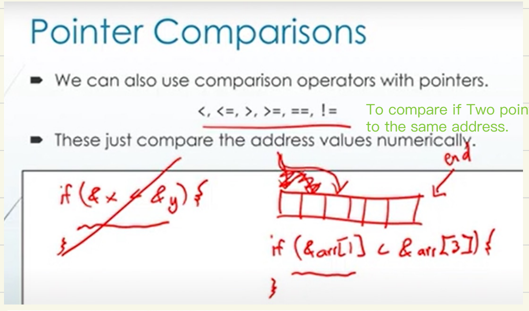
Trace this code and draw a memory diagram as you go. Once you're finished, use your diagram to answer the question below.

```
int main() {
  int arr[5] = {6, 3, 2, 4, 5};
  int *a = arr;
  int *b = arr + 2;
  int *c = b + 1;
  int *d = &arr[1];

  ++a;
  --b;
  c = d;
  c += 2;

  cout << *a << endl;
  cout << *(a + 2) << endl;
  cout << (a - d) << endl;
  cout << (b - c) << endl;
  cout << b[2] << endl;
  cout << *(arr+5) << endl;
}
```

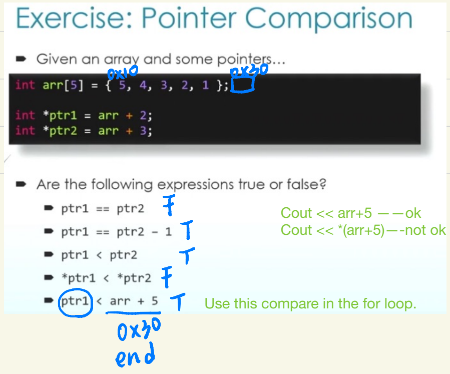
vedio three: pointer comparisons



Pointer Comparisons

- We can also use comparison operators with pointers.
  - <, <=, >, >=, ==, !=
- These just compare the address values numerically.

To compare if Two pointers points to the same address.



Exercise: Pointer Comparison

- Given an array and some pointers…

```
int arr[5] = { 5, 4, 3, 2, 1 };

int *ptr1 = arr + 2;
int *ptr2 = arr + 3;
```

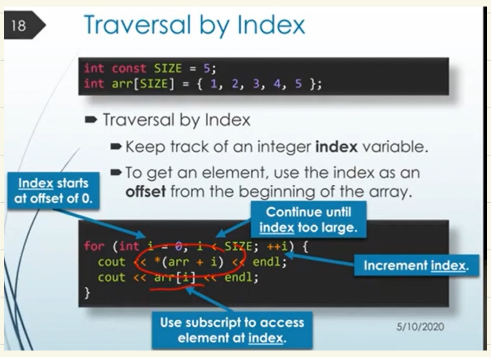
- Are the following expressions true or false?
  - `ptr1 == ptr2` F
  - `ptr1 == ptr2 - 1` T
  - `ptr1 < ptr2` T
  - `*ptr1 < *ptr2` F
  - `ptr1 < arr + 5` T

Cout << arr+5 ——ok

Cout << *(arr+5)—-not ok

Use this compare in the for loop.

Video four: traversal by pointer



18 Traversal by Index

```
int const SIZE = 5;
int arr[SIZE] = { 1, 2, 3, 4, 5 };
```

- Traversal by Index
  - Keep track of an integer **index** variable.
  - To get an element, use the index as an **offset** from the beginning of the array.

```
for (int i = 0; i < SIZE; ++i) {
  cout << *(arr + i) << endl;
  cout << arr[i] << endl;
}
```

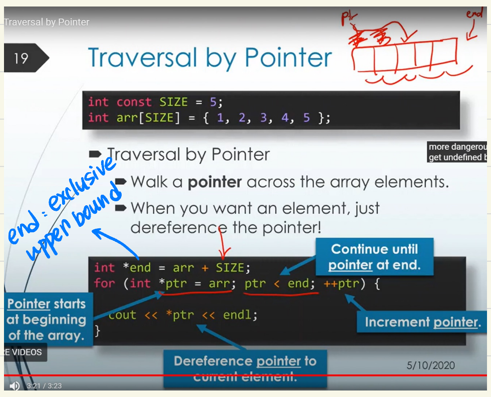
Index starts at offset of 0.

Continue until index too large.

Increment index.

Use subscript to access element at index.

5/10/2020



Traversal by Pointer

19 Traversal by Pointer

```
int const SIZE = 5;
int arr[SIZE] = { 1, 2, 3, 4, 5 };
```

- Traversal by Pointer
  - Walk a **pointer** across the array elements.
  - When you want an element, just dereference the pointer!

```
int *end = arr + SIZE;
for (int *ptr = arr; ptr < end; ++ptr) {
  cout << *ptr << endl;
}
```

Continue until pointer at end.

Pointer starts at beginning of the array.

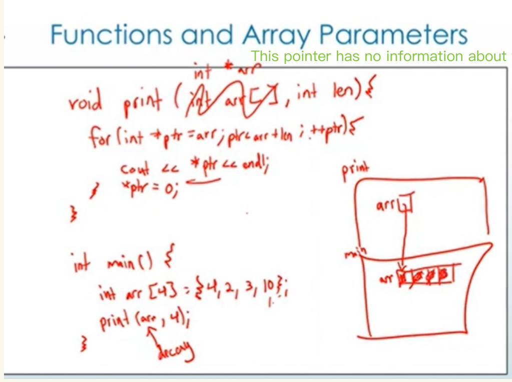
Increment pointer.

Dereference pointer to current element.

more dangerou get undefined b

5/10/2020

Video five: array parameters and tunctioons



Functions and Array Parameters

This pointer has no information about the size of the array

int *arr
void print (int arr[4], int len){
for (int *ptr = arr; ptr < arr+len ; ++ptr){
cout << *ptr << endl;
*ptr = 0;
}
}
int main() {
int arr [4] = {4, 2, 3, 10};
print (arr, 4);
}
decay
print
arr
main
arr

Lecture recording notes